

master ▾

...

noisy-networks-measurements / examples / readme.md

jg-you Restructured dir

🕒 History

👤 1 contributor

Raw Blame



21 lines (11 sloc) 2.12 KB

## Template list

### Models obtained by modifying the `undir.stan` template

- `poisson_data_ER_prior.stan`: Takes a symmetric matrix of number of observations  $x$  as input. Models  $x[i,j]$  as drawn from a Poisson variable, with success rate determined by presence or absence of the edge  $(i,j)$ . Uses an Erdős-Rényi prior.
- `bernoulli_data_ER_prior.stan`: Takes as input a symmetric matrix of number of observations  $x$ , and a total number of observations  $m$ . Models  $x[i,j]$  as the outcome of  $m$  draws, with success rate determined by presence or absence of the edge  $(i,j)$ . Uses an Erdős-Rényi prior.
- `poisson_data_SCM_prior.stan`: Same as Poisson data model above, but uses a soft configuration model as the network prior.
- `bernoulli_data_SCM_prior.stan`: Same as Bernoulli data model above, but uses a soft configuration model as the network prior.

### Models obtained by modifying the `undir_multitype.stan` template

- `multitype_poisson_data_ER_prior.stan`: Takes a symmetric matrix of number of observations  $x$  as input. Models  $x[i,j]$  as drawn from a Poisson variable, with success rate determined by presence or absence of the edge  $(i,j)$ . Uses multiple edge types. Uses a generalized Erdős-Rényi prior.

### Accelerated versions (no templates)

- `fast_poisson.stan`: Accelerated version of `poisson.stan`. Takes two vectors  $x$  and  $y$  as input. The entry  $x[i]$  of  $x$  gives the number of pairs of nodes that are measured as interacting  $y[i]$  times. Return the matrix  $Q$  in vector format matching  $x$  and  $y$ .
- `fast_bernoulli.stan`: Accelerated version of `bernoulli.stan`. Takes as input two vectors  $x$  and  $y$ , and a total number of observations  $m$ . The entry  $x[i]$  of  $x$  gives the number of pairs of nodes that are measured as interacting  $y[i]$  times, out of  $m$  measurements. Return the matrix  $Q$  in vector format matching  $x$  and  $y$ .

master ▾

...

noisy-networks-measurements / examples / **bernoulli\_data\_ER\_prior.stan**

jg-you Restructured dir

🕒 History

1 contributor

Raw Blame



48 lines (43 sloc) 1.29 KB

```
1 data {
2   int<lower=1> n;
3   int<lower=0> X[n, n];
4   int M;
5   real<lower=0> rates_prior[2];
6   real<lower=0> rho_prior[2];
7 }
8 parameters {
9   positive_ordered[2] rates;
10  real<lower=0, upper=1> rho;
11 }
12 model {
13   rates ~ beta(rates_prior[1], rates_prior[2]);
14   rho ~ beta(rho_prior[1], rho_prior[2]);
15
16   for (i in 1:n) {
17     for (j in i + 1:n) {
18       real log_mu_ij_0 = binomial_lpmf(X[i, j] | M, rates[1]);
19       real log_mu_ij_1 = binomial_lpmf(X[i, j] | M, rates[2]);
20
21       real log_nu_ij_0 = bernoulli_lpmf(0 | rho);
22       real log_nu_ij_1 = bernoulli_lpmf(1 | rho);
23
24       real z_ij_0 = log_mu_ij_0 + log_nu_ij_0;
25       real z_ij_1 = log_mu_ij_1 + log_nu_ij_1;
26       if (z_ij_0 > z_ij_1) {target += z_ij_0 + log1p_exp(z_ij_1 - z_ij_0);}
27       else {target += z_ij_1 + log1p_exp(z_ij_0 - z_ij_1);}
28     }
29   }
30 }
31 generated quantities {
32   real Q[n, n];
33   for (i in 1:n) {
34     Q[i, i] = 0;
35     for (j in i+1:n) {
36       real log_mu_ij_0 = binomial_lpmf(X[i, j] | M, rates[1]);
37       real log_mu_ij_1 = binomial_lpmf(X[i, j] | M, rates[2]);
38
39       real log_nu_ij_0 = bernoulli_lpmf(0 | rho);
40       real log_nu_ij_1 = bernoulli_lpmf(1 | rho);
41
42       real z_ij_0 = log_mu_ij_0 + log_nu_ij_0;
43       real z_ij_1 = log_mu_ij_1 + log_nu_ij_1;
44       Q[i, j] = 1 / (1 + exp(z_ij_0 - z_ij_1));
45       Q[j, i] = Q[i, j];
46     }
47   }
48 }
```

master

...

noisy-networks-measurements / examples / **bernoulli\_data\_SCM\_prior.stan**

jg-you Remove unused parameter from models

History

1 contributor

Raw Blame



50 lines (45 sloc) 1.36 KB

```
1 data {
2   int<lower=1> n;
3   int<lower=0> X[n, n];
4   int M;
5   real<lower=0> rates_prior[2];
6 }
7 parameters {
8   positive_ordered[2] rates;
9   simplex[n] lambda;
10  real<lower = 0> scale;
11 }
12 model {
13   rates ~ beta(rates_prior[1], rates_prior[2]);
14   scale ~ exponential(100);
15
16   for (i in 1:n) {
17     for (j in 1:n) {
18       real log_mu_ij_0 = binomial_lpmf(X[i, j] | M, rates[1]);
19       real log_mu_ij_1 = binomial_lpmf(X[i, j] | M, rates[2]);
20
21       real r = inv_logit(scale * lambda[i] * lambda[j]);
22       real log_nu_ij_0 = bernoulli_lpmf(0 | r);
23       real log_nu_ij_1 = bernoulli_lpmf(1 | r);
24
25       real z_ij_0 = log_mu_ij_0 + log_nu_ij_0;
26       real z_ij_1 = log_mu_ij_1 + log_nu_ij_1;
27       if (z_ij_0 > z_ij_1) {target += z_ij_0 + log1p_exp(z_ij_1 - z_ij_0);}
28       else {target += z_ij_1 + log1p_exp(z_ij_0 - z_ij_1);}
29     }
30   }
31 }
32 generated quantities {
33   real Q[n, n];
34   for (i in 1:n) {
35     Q[i, 1] = 0;
36     for (j in i+1:n) {
37       real log_mu_ij_0 = binomial_lpmf(X[i, j] | M, rates[1]);
38       real log_mu_ij_1 = binomial_lpmf(X[i, j] | M, rates[2]);
39
40       real r = inv_logit(scale * lambda[i] * lambda[j]);
41       real log_nu_ij_0 = bernoulli_lpmf(0 | r);
42       real log_nu_ij_1 = bernoulli_lpmf(1 | r);
43
44       real z_ij_0 = log_mu_ij_0 + log_nu_ij_0;
45       real z_ij_1 = log_mu_ij_1 + log_nu_ij_1;
46       Q[i, j] = 1 / (1 + exp(z_ij_0 - z_ij_1));
47       Q[j, i] = Q[i, j];
48     }
49   }
50 }
```

master ▾

...

noisy-networks-measurements / examples / poisson\_data\_ER\_prior.stan

jg-you Restructured dir

🕒 History

1 contributor

Raw

Blame



48 lines (43 sloc) 1.31 KB

```
1 data {
2   int<lower=1> n;
3   int<lower=0> X[n, n];
4   real<lower=0> rates_std_prior[2];
5   real<lower=0> rho_prior[2];
6 }
7 parameters {
8   positive_ordered[2] rates;
9   real<lower=0, upper=1> rho;
10 }
11 model {
12   rates[1] ~ normal(1, rates_std_prior[1]);
13   rates[2] ~ normal(1, rates_std_prior[2]);
14   rho ~ beta(rho_prior[1], rho_prior[2]);
15
16   for (i in 1:n) {
17     for (j in i + 1:n) {
18       real log_mu_ij_0 = poisson_lpmf(X[i, j] | rates[1]);
19       real log_mu_ij_1 = poisson_lpmf(X[i, j] | rates[2]);
20
21       real log_nu_ij_0 = bernoulli_lpmf(0 | rho);
22       real log_nu_ij_1 = bernoulli_lpmf(1 | rho);
23
24       real z_ij_0 = log_mu_ij_0 + log_nu_ij_0;
25       real z_ij_1 = log_mu_ij_1 + log_nu_ij_1;
26       if (z_ij_0 > z_ij_1) {target += z_ij_0 + log1p_exp(z_ij_1 - z_ij_0);}
27       else {target += z_ij_1 + log1p_exp(z_ij_0 - z_ij_1);}
28     }
29   }
30 }
31 generated quantities {
32   real Q[n, n];
33   for (i in 1:n) {
34     Q[i, i] = 0;
35     for (j in i+1:n) {
36       real log_mu_ij_0 = poisson_lpmf(X[i, j] | rates[1]);
37       real log_mu_ij_1 = poisson_lpmf(X[i, j] | rates[2]);
38
39       real log_nu_ij_0 = bernoulli_lpmf(0 | rho);
40       real log_nu_ij_1 = bernoulli_lpmf(1 | rho);
41
42       real z_ij_0 = log_mu_ij_0 + log_nu_ij_0;
43       real z_ij_1 = log_mu_ij_1 + log_nu_ij_1;
44       Q[i, j] = 1 / (1 + exp(z_ij_0 - z_ij_1));
45       Q[j, i] = Q[i, j];
46     }
47   }
48 }
```

master ▾

...

noisy-networks-measurements / examples / poisson\_data\_SCM\_prior.stan

jg-you Remove unused parameter from models

History

1 contributor

Raw Blame



50 lines (45 sloc) 1.38 KB

```
1 data {
2   int<lower=1> n;
3   int<lower=0> X[n, n];
4   real<lower=0> rates_std_prior[2];
5 }
6 parameters {
7   positive_ordered[2] rates;
8   simplex[n] lambda;
9   real<lower = 0> scale;
10 }
11 model {
12   rates[1] ~ normal(1, rates_std_prior[1]);
13   rates[2] ~ normal(1, rates_std_prior[2]);
14   scale ~ exponential(100);
15
16   for (i in 1:n) {
17     for (j in 1 + 1:n) {
18       real log_mu_ij_0 = poisson_lpmf(X[i, j] | rates[1]);
19       real log_mu_ij_1 = poisson_lpmf(X[i, j] | rates[2]);
20
21       real r = inv_logit(scale * lambda[i] * lambda[j]);
22       real log_nu_ij_0 = bernoulli_lpmf(0 | r);
23       real log_nu_ij_1 = bernoulli_lpmf(1 | r);
24
25       real z_ij_0 = log_mu_ij_0 + log_nu_ij_0;
26       real z_ij_1 = log_mu_ij_1 + log_nu_ij_1;
27       if (z_ij_0 > z_ij_1) {target += z_ij_0 + log1p_exp(z_ij_1 - z_ij_0);}
28       else {target += z_ij_1 + log1p_exp(z_ij_0 - z_ij_1);}
29     }
30   }
31 }
32 generated quantities {
33   real Q[n, n];
34   for (i in 1:n) {
35     Q[i, i] = 0;
36     for (j in i+1:n) {
37       real log_mu_ij_0 = poisson_lpmf(X[i, j] | rates[1]);
38       real log_mu_ij_1 = poisson_lpmf(X[i, j] | rates[2]);
39
40       real r = inv_logit(scale * lambda[i] * lambda[j]);
41       real log_nu_ij_0 = bernoulli_lpmf(0 | r);
42       real log_nu_ij_1 = bernoulli_lpmf(1 | r);
43
44       real z_ij_0 = log_mu_ij_0 + log_nu_ij_0;
45       real z_ij_1 = log_mu_ij_1 + log_nu_ij_1;
46       Q[i, j] = 1 / (1 + exp(z_ij_0 - z_ij_1));
47       Q[j, i] = Q[i, j];
48     }
49   }
50 }
```



master ▾

...

noisy-networks-measurements / examples / multitype\_poisson\_data\_ER\_prior.stan

jg-you Restructured dir

🕒 History

1 contributor

Raw Blame



62 lines (59 sloc) 1.25 KB

```
1 data {
2   int<lower=1> n;
3   int<lower=0> X[n, n];
4   int T; // number of edge types
5   real<lower=0> rates_std_prior[T];
6 }
7 parameters {
8   positive_ordered[T] rates;
9   simplex[T] rho;
10 }
11 model {
12   for (k in 1:T)
13   {
14     rates[k] ~ normal(1, rates_std_prior[k]);
15   }
16
17   for (i in 1:n) {
18     for (j in i + 1:n) {
19       vector[T] z_ij;
20       vector[T] z_max_vector;
21       real z_max;
22       for (k in 1:T) {
23         real log_mu_ij_k = poisson_lpmf(X[i, j] | rates[k]);
24         real log_nu_ij_k = log(rho[k]);
25
26         z_ij[k] = log_mu_ij_k + log_nu_ij_k;
27       }
28       z_max = max(z_ij);
29       z_max_vector = rep_vector(z_max, T);
30       target += z_max + log_sum_exp(z_ij - z_max_vector);
31     }
32   }
33 }
34 generated quantities {
35   real Q[n, n, T];
36   for (i in 1:n) {
37     for (k in 1:T) {
38       Q[i, i, k] = 0;
39     }
40     for (j in i+1:n) {
41       vector[T] z_ij;
42       real accu;
43       for (k in 1:T)
44       {
45         real log_mu_ij_k = poisson_lpmf(X[i, j] | rates[k]);
46         real log_nu_ij_k = log(rho[k]);
47
48         z_ij[k] = log_mu_ij_k + log_nu_ij_k;
49       }
50       for (k in 1:T)
51       {
52         accu = 0;
53         for (k_prime in 1:T)
54         {
55           accu += exp(z_ij[k_prime] - z_ij[k]);
56         }
57         Q[i, j, k] = 1 / accu;
58         Q[j, i, k] = Q[i, j, k];
59       }
60     }
61   }
62 }
```

master

...

noisy-networks-measurements / examples / compiler.py

jg-you Update email

History

1 contributor

Raw Blame



48 lines (41 sloc) 1.64 KB

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Python functions to interact with the stan models.
5
6  Author:
7  Jean-Gabriel Young <jean.gabriel.young@gmail.com>
8  """
9  import numpy as np
10 import pickle
11 import pystan
12 import os
13
14 abs_path = os.path.dirname(os.path.abspath(__file__))
15
16
17 def compile_stan_model(model_file, force=False):
18     """Autocompile Stan model."""
19     model_name = os.path.splitext(model_file)[0]
20     source_path = os.path.join(abs_path, model_name + '.stan')
21     target_path = os.path.join(abs_path, model_name + '.bin')
22
23     if os.path.exists(target_path):
24         # Test whether the model has changed and only compile if it did
25         with open(target_path, 'rb') as f:
26             current_model = pickle.load(f)
27         with open(source_path, 'r') as f:
28             file_content = "".join([line for line in f])
29         if file_content != current_model.model_code or force:
30             print(target_path, "[Compiling]", ["", "[Forced]"][force])
31             model = pystan.StanModel(source_path, model_name=model_name)
32             with open(target_path, 'wb') as f:
33                 pickle.dump(model, f)
34         else:
35             print(target_path, "[Skipping --- already compiled]")
36     else:
37         # If model binary does not exist, compile it
38         print(target_path, "[Compiling]")
39         model = pystan.StanModel(source_path, model_name=model_name)
40         with open(target_path, 'wb') as f:
41             pickle.dump(model, f)
42
43
44 if __name__ == '__main__':
45     print("Compiling models with pystan version", pystan.__version__)
46     for m in os.listdir("examples"):
47         if os.path.splitext(m)[1] == ".stan":
48             compile_stan_model(m, force=False)

```

master ▾

...

noisy-networks-measurements / examples / fast\_bernoulli.stan

jg-you Typos

🕒 History

👤 1 contributor

Raw

Blame



47 lines (42 sloc) 1.3 KB

```
1 data {
2   int n; // number of non-empty pairs class
3   int X[n]; // number of pairs of nodes in each classes
4   int Y[n]; // number of observations for each class
5   int M; // Total number of interactions
6   // Priors
7   real<lower=0> rates_prior[2];
8   real<lower=0> rho_prior[2];
9 }
10 parameters {
11   positive_ordered[2] rates;
12   real<lower=0, upper=1> rho;
13 }
14 model {
15   rates ~ beta(rates_prior[1], rates_prior[2]);
16   rho ~ beta(rho_prior[1], rho_prior[2]);
17
18   for (i in 1:n)
19   {
20
21     real log_mu_i_0 = binomial_lpmf(Y[i] | M, rates[1]);
22     real log_mu_i_1 = binomial_lpmf(Y[i] | M, rates[2]);
23     real log_nu_i_0 = bernoulli_lpmf(0 | rho);
24     real log_nu_i_1 = bernoulli_lpmf(1 | rho);
25
26     real z = 0;
27     real z_i_0 = log_mu_i_0 + log_nu_i_0;
28     real z_i_1 = log_mu_i_1 + log_nu_i_1;
29     if (z_i_0 > z_i_1) {z += z_i_0 + log1p_exp(z_i_1 - z_i_0);}
30     else {z += z_i_1 + log1p_exp(z_i_0 - z_i_1);}
31     target += X[i] * z;
32   }
33 }
34 generated quantities {
35   real Q[n];
36   for (i in 1:n) {
37     real log_mu_i_0 = binomial_lpmf(Y[i] | M, rates[1]);
38     real log_mu_i_1 = binomial_lpmf(Y[i] | M, rates[2]);
39     real log_nu_i_0 = bernoulli_lpmf(0 | rho);
40     real log_nu_i_1 = bernoulli_lpmf(1 | rho);
41     real z_i_0 = log_mu_i_0 + log_nu_i_0;
42     real z_i_1 = log_mu_i_1 + log_nu_i_1;
43     Q[i] = 1 / (1 + exp(z_i_1 - z_i_0));
44   }
45 }
46
47
```



🔑 master ▾

...

noisy-networks-measurements / examples / fast\_poisson.stan

👤 jg-you Typos

🕒 History

👤 1 contributor

Raw Blame




47 lines (42 sloc) 1.28 KB

```
1 data {
2   int n; // number of non-empty pairs class
3   int X[n]; // number of pairs of nodes in each classes
4   int Y[n]; // number of observations for each class
5   // Priors
6   real<lower=0> rates_std_prior[2];
7   real<lower=0> rho_prior[2];
8 }
9 parameters {
10  positive_ordered[2] rates;
11  real<lower=0, upper=1> rho;
12 }
13 model {
14  rates[1] ~ normal(1, rates_std_prior[1]);
15  rates[2] ~ normal(1, rates_std_prior[2]);
16  rho ~ beta(rho_prior[1], rho_prior[2]);
17
18  for (i in 1:n)
19  {
20
21    real log_mu_i_0 = poisson_lpmf(Y[i] | rates[1]);
22    real log_mu_i_1 = poisson_lpmf(Y[i] | rates[2]);
23    real log_nu_i_0 = bernoulli_lpmf(0 | rho);
24    real log_nu_i_1 = bernoulli_lpmf(1 | rho);
25
26    real z = 0;
27    real z_i_0 = log_mu_i_0 + log_nu_i_0;
28    real z_i_1 = log_mu_i_1 + log_nu_i_1;
29    if (z_i_0 > z_i_1) {z += z_i_0 + log1p_exp(z_i_1 - z_i_0);}
30    else {z += z_i_1 + log1p_exp(z_i_0 - z_i_1);}
31    target += X[i] * z;
32  }
33 }
34 generated quantities {
35  real Q[n];
36  for (i in 1:n) {
37    real log_mu_i_0 = poisson_lpmf(Y[i] | rates[1]);
38    real log_mu_i_1 = poisson_lpmf(Y[i] | rates[2]);
39    real log_nu_i_0 = bernoulli_lpmf(0 | rho);
40    real log_nu_i_1 = bernoulli_lpmf(1 | rho);
41    real z_i_0 = log_mu_i_0 + log_nu_i_0;
42    real z_i_1 = log_mu_i_1 + log_nu_i_1;
43    Q[i] = 1 / (1 + exp(z_i_1 - z_i_0));
44  }
45 }
46
47
```

master ▾

...

noisy-networks-measurements / templates / **readme.md** jg-you Typo History

👤 1 contributor

Raw Blame

9 lines (7 sloc) 877 Bytes

Model templates, that can be used to implement custom models without writing boilerplate code. Portions of the code that must be modified are marked with double square brackets, like so: `[[ ]]`. The available templates are:

- `undir.stan` : Generic template for measurement of undirected graph.
- `dir.stan` : Generic template for measurement of directed graph.
- `undir_multitype.stan` : Generic template for measurement of undirected graph with many edge types.

See `stan` reference manual for details on how to use various probability mass functions and probability density functions.

master ▾

...

noisy-networks-measurements / templates / dir.stan

jg-you Restructured dir

History

1 contributor

Raw Blame



60 lines (55 sloc) 1.4 KB

```
1 data {
2   int<lower=1> n;
3   int<lower=0> X[n, n];
4   // [[Additional data go here]]
5 }
6 parameters {
7   // [[Parameters go here]]
8   //
9   // For example:
10  // real<lower=0,upper=1> theta;
11 }
12 model {
13   // [[Priors go here]]
14   //
15   // For example:
16   // theta ~ beta(1, 1/2);
17
18   for (i in 1:n) {
19     for (j in 1:n) {
20       if (i != j) {
21         // [[Data model goes here]]
22         real log_mu_ij_0 = ;
23         real log_mu_ij_1 = ;
24
25         // [[Network model goes here]]
26         real log_nu_ij_0 = ;
27         real log_nu_ij_1 = ;
28
29         // Boilerplate code below, do not modify
30         real z_ij_0 = log_mu_ij_0 + log_nu_ij_0;
31         real z_ij_1 = log_mu_ij_1 + log_nu_ij_1;
32         if (z_ij_0 > z_ij_1) {target += z_ij_0 + log1p_exp(z_ij_1 - z_ij_0);}
33         else {target += z_ij_1 + log1p_exp(z_ij_0 - z_ij_1);}
34       }
35     }
36   }
37 }
38 generated quantities {
39   // Generate edge probability matrix
40   real Q[n, n];
41   for (i in 1:n) {
42     Q[i, i] = 0;
43     for (j in 1:n) {
44       if (i != j) {
45         // [[Data model goes here, as in model block]]
46         real log_mu_ij_0 = ;
47         real log_mu_ij_1 = ;
48
49         // [[Network model goes here, as in model block]]
50         real log_nu_ij_0 = ;
51         real log_nu_ij_1 = ;
52
53         // Boilerplate code below, do not modify
54         real z_ij_0 = log_mu_ij_0 + log_nu_ij_0;
55         real z_ij_1 = log_mu_ij_1 + log_nu_ij_1;
56         Q[i, j] = 1 / (1 + exp(z_ij_0 - z_ij_1));
57       }
58     }
59   }
60 }
```

master ▾

...

noisy-networks-measurements / templates / **undir.stan**

jg-you Restructured dir

🕒 History

👤 1 contributor

Raw

Blame



57 lines (52 sloc) 1.34 KB

```
1 data {
2   int<lower=1> n;
3   int<lower=0> X[n, n];
4   // [[Additional data go here]]
5 }
6 parameters {
7   // [[Parameters go here]]
8   //
9   // For example:
10  // real<lower=0,upper=1> theta;
11 }
12 model {
13   // [[Priors go here]]
14   //
15   // For example:
16   // theta ~ beta(1, 1/2);
17
18   for (i in 1:n) {
19     for (j in i + 1:n) {
20       // [[Data model goes here]]
21       real log_mu_ij_0 = ;
22       real log_mu_ij_1 = ;
23
24       // [[Network model goes here]]
25       real log_nu_ij_0 = ;
26       real log_nu_ij_1 = ;
27
28       // Boilerplate code below, do not modify
29       real z_ij_0 = log_mu_ij_0 + log_nu_ij_0;
30       real z_ij_1 = log_mu_ij_1 + log_nu_ij_1;
31       if (z_ij_0 > z_ij_1) {target += z_ij_0 + log1p_exp(z_ij_1 - z_ij_0);}
32       else {target += z_ij_1 + log1p_exp(z_ij_0 - z_ij_1);}
33     }
34   }
35 }
36 generated quantities {
37   // Generate edge probability matrix
38   real Q[n, n];
39   for (i in 1:n) {
40     Q[i, i] = 0;
41     for (j in i+1:n) {
42       // [[Data model goes here, as in model block]]
43       real log_mu_ij_0 = ;
44       real log_mu_ij_1 = ;
45
46       // [[Network model goes here, as in model block]]
47       real log_nu_ij_0 = ;
48       real log_nu_ij_1 = ;
49
50       // Boilerplate code below, do not modify
51       real z_ij_0 = log_mu_ij_0 + log_nu_ij_0;
52       real z_ij_1 = log_mu_ij_1 + log_nu_ij_1;
53       Q[i, j] = 1 / (1 + exp(z_ij_0 - z_ij_1));
54       Q[j, i] = Q[i, j];
55     }
56   }
57 }
```

Raw Blame



70 lines (67 sloc) 1.46 KB

```
1 data {
2   int<lower=1> n;
3   int<lower=0> X[n, n];
4   int T; // number of edge types
5   // [[Additional data go here]]
6 }
7 parameters {
8   // [[Parameters go here]]
9   //
10  // For example:
11  // real<lower=0,upper=1> theta;
12 }
13 model {
14   // [[Priors go here]]
15   //
16   // For example:
17   // theta ~ beta(1, 1/2);
18
19   for (i in 1:n) {
20     for (j in i + 1:n) {
21       vector[T] z_ij;
22       vector[T] z_max_vector;
23       real z_max;
24       for (k in 1:T) {
25         // [[Data model goes here]]
26         real log_mu_ij_k = ;
27         // [[Network model goes here]]
28         real log_nu_ij_k = ;
29
30         // Boilerplate code below, do not modify
31         z_ij[k] = log_mu_ij_k + log_nu_ij_k;
32       }
33       z_max = max(z_ij);
34       z_max_vector = rep_vector(z_max, T);
35       target += z_max + log_sum_exp(z_ij - z_max_vector);
36     }
37   }
38 }
39 generated quantities {
40   // Generate edge probability matrix
41   real Q[n, n, T];
42   for (i in 1:n) {
43     for (k in 1:T) {
44       Q[i, i, k] = 0;
45     }
46     for (j in i+1:n) {
47       vector[T] z_ij;
48       real accu;
49       for (k in 1:T)
50       {
51         // [[Data model goes here]]
52         real log_mu_ij_k = ;
53         // [[Network model goes here]]
54         real log_nu_ij_k = ;
55
56         // Boilerplate code below, do not modify
57         z_ij[k] = log_mu_ij_k + log_nu_ij_k;
58       }
59       for (k in 1:T)
60       {
61         accu = 0;
62         for (k_prime in 1:T)
63         {
64           accu += exp(z_ij[k_prime] - z_ij[k]);
65         }
66         Q[i, j, k] = 1 / accu;
67       }
68     }
69   }
70 }
```